In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

In [2]:
```
data = pd.read_csv('diabetes.csv')
```

In [3]:
```
data
```

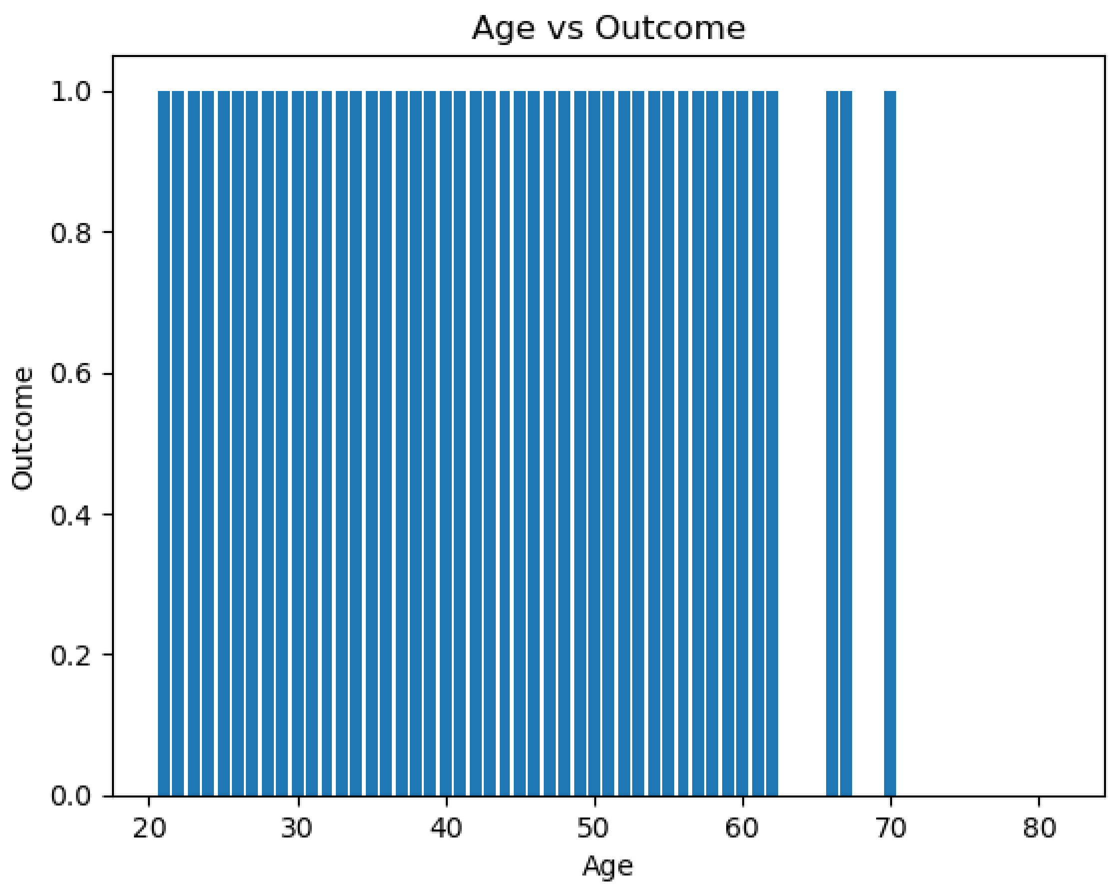
Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

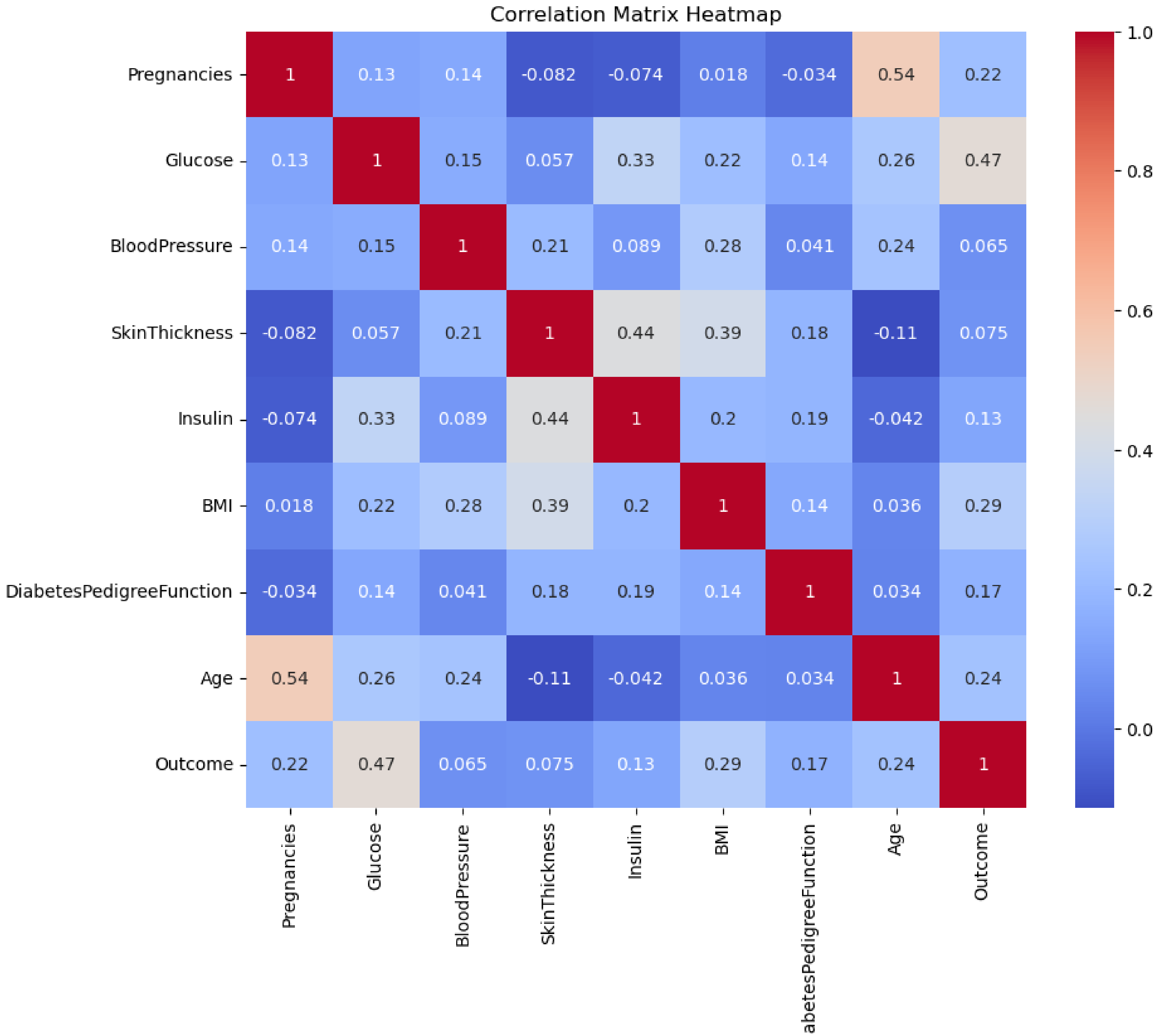
In [4]:

```
# Create a bar graph of Age vs Outcome
plt.bar(data['Age'], data['Outcome'])

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Outcome')
plt.title('Age vs Outcome')

# Display the plot
plt.show()
```



In [8]:

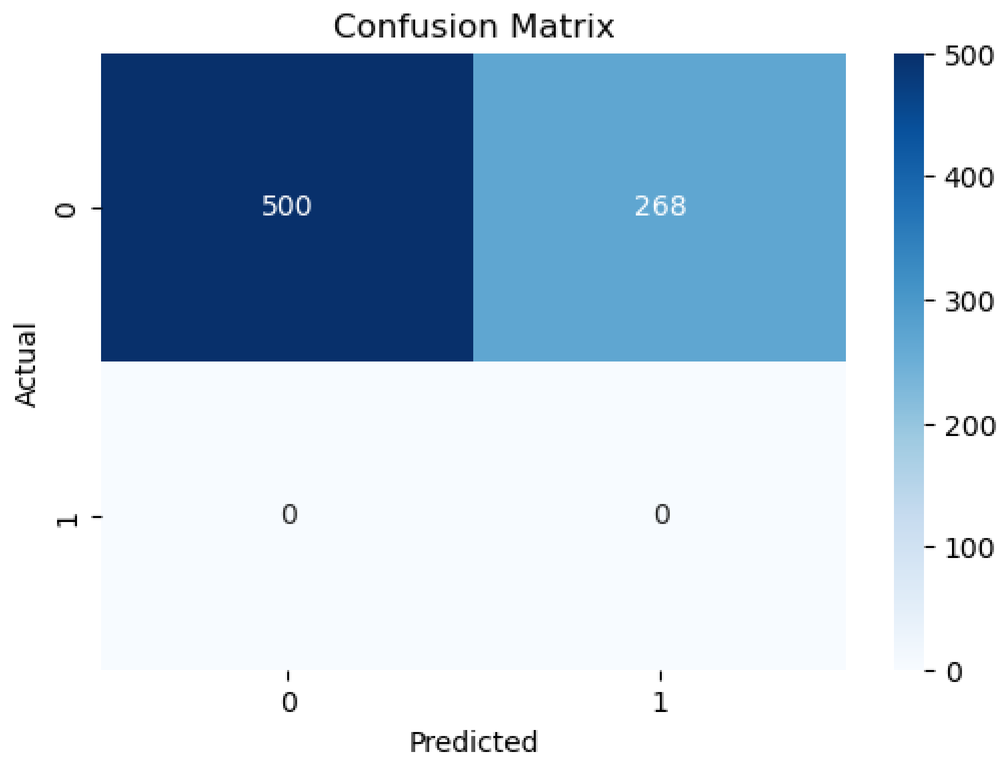
```
import seaborn as sns
# Calculate the correlation matrix
corr_matrix = data.corr()

# Plot the correlation matrix heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

In [11]:

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Extract the 'Outcome' column as the predicted labels
outcome = data['Outcome'].values

# Create an array of the same length with the true labels (assuming all are 0)
true_labels = np.zeros(len(outcome))

# Calculate the confusion matrix
cm = confusion_matrix(true_labels, outcome)

# Create a DataFrame for the confusion matrix
cm_df = pd.DataFrame(cm, index=['Actual 0', 'Actual 1'], columns=['Predicted 0', 'Predicted 1'])

# Create a heatmap using seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues')

# Add labels, title, and ticks
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.xticks([0.5, 1.5], ['0', '1'])
plt.yticks([0.5, 1.5], ['0', '1'])
plt.show()


# This is still empty confusion matrix
```



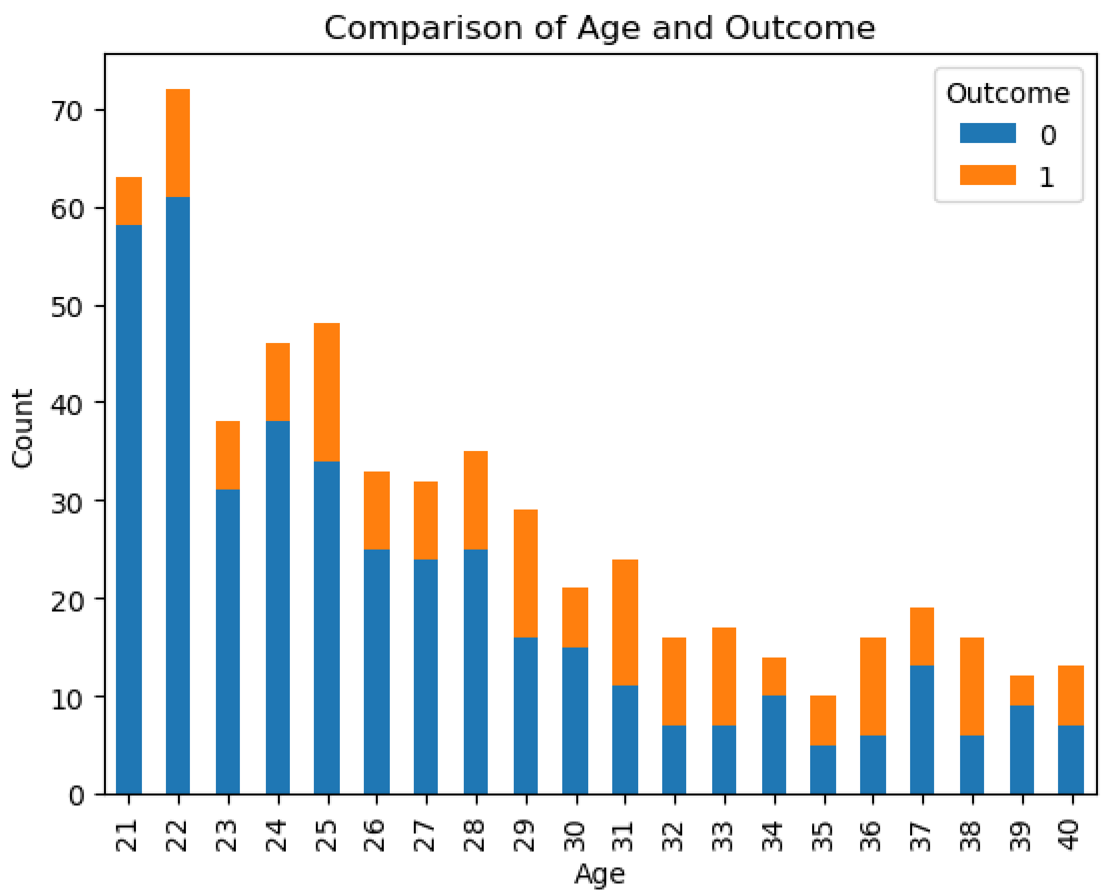
In [12]:

```
# Use crosstab to compare 'Age' and 'Outcome' columns
cross_tab = pd.crosstab(data['Age'], data['Outcome'])

print(cross_tab[:10])
```

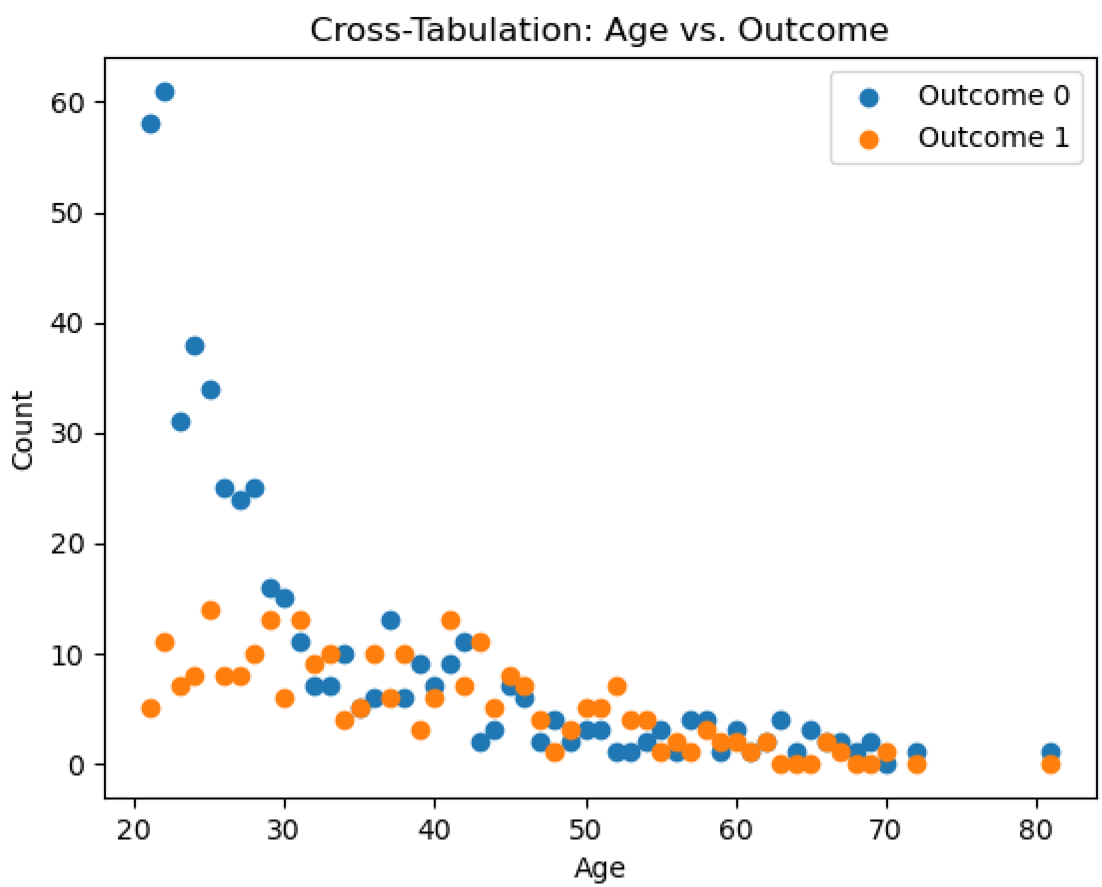
```
Outcome   0   1
Age
21       58   5
22       61  11
23       31   7
24       38   8
25       34  14
26       25   8
27       24   8
28       25  10
29       16  13
30       15   6
```

In [13]:

```
# Plotting the bar chart
cross_tab[:20].plot(kind='bar', stacked=True)

# Adding labels and title
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Comparison of Age and Outcome')

# Display the plot
plt.show()
```



In [14]:
```
# Use crosstab to compare 'Age' and 'Outcome' columns
cross_tab = pd.crosstab(data['Age'], data['Outcome'])

# Visualize the cross-tabulation as scatter plots
for outcome in cross_tab.columns:
    plt.scatter(cross_tab.index, cross_tab[outcome], label=f'Outcome {outcome}')

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Cross-Tabulation: Age vs. Outcome')
plt.legend()

# Display the plot
plt.show()
```



In [15]:
```
data.isna().sum()
```

Out[15]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [16]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [17]:
```
data.columns
```

Out[17]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [18]:
```
data.shape
```

Out[18]:
```
(768, 9)
```

In [19]:
```
# Getting the statistical measures of the data
data.describe()
```

Out[19]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.34
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.47
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.00
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.00
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.00
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.00
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.00

In [20]:
```
data['Outcome'].value_counts()
```

Out[20]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

0 = Non-Diabetic

1 = Diabetic

In [21]:
```
data.groupby('Outcome').mean()
```

Out[21]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

splitting our data

In [22]:
```
x = data.drop('Outcome', axis=1)
y = data['Outcome']
```

In [23]:
```
x.shape, y.shape
```

Out[23]:
```
((768, 8), (768,))
```

In [24]:
```
x
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

In [25]:
```
y
```

Out[25]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

In [26]:
```
scalar = StandardScaler()
scalar.fit(x)

```

Out[26]:
```
StandardScaler()
```

In [27]:
```
standardized_data = scalar.transform(x)
```

In [28]:
```
standardized_data
```

Out[28]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

In [29]:
```
x = standardized_data
y = data['Outcome']
```

In [30]:
```
x, y
```

Out[30]:
```
(array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
          0.46849198,  1.4259954 ],
        [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
         -0.36506078, -0.19067191],
        [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
          0.60439732, -0.10558415],
        ...,
        [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
         -0.68519336, -0.27575966],
        [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
         -0.37110101,  1.17073215],
        [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
         -0.47378505, -0.87137393]]),
 0      1
 1      0
 2      1
 3      0
 4      1
       ..
 763    0
 764    0
 765    0
 766    1
 767    0
 Name: Outcome, Length: 768, dtype: int64)
```

Train Test Split

In [31]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_state=2)
```

In [32]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[32]:
```
((614, 8), (154, 8), (614,), (154,))
```

Training Model

In [33]:
```
model = svm.SVC(kernel='linear')
```

In [34]:
```
model.fit(x_train, y_train)
```

Out[34]:
```
SVC(kernel='linear')
```

Model evaluation

1

In [35]:
```
# Assuming you have obtained the predictions using model.predict(x_test)
from sklearn import metrics
y_pred = model.predict(x_test)

# Compute accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Compute precision, recall, and F1-score
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
f1_score = metrics.f1_score(y_test, y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1_score)
```

```
Accuracy: 0.7727272727272727
Precision: 0.7567567567567568
Recall: 0.5185185185185185
F1-Score: 0.6153846153846154
```

2

In [36]:

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': [0.1, 1, 10]
}

# Create the SVM model
model = svm.SVC()

# Create the GridSearchCV object
grid_search = GridSearchCV(model, param_grid, scoring='accuracy', cv=5)

# Perform grid search to find the best hyperparameters
grid_search.fit(x_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)

# Evaluate the best model
accuracy = metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
f1_score = metrics.f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1_score)

```

```
Best Hyperparameters: {'C': 1, 'gamma': 0.1, 'kernel': 'linear'}
Accuracy: 0.7727272727272727
Precision: 0.7567567567567568
Recall: 0.5185185185185185
F1-Score: 0.6153846153846154
```

In [37]:
```
y_pred
```

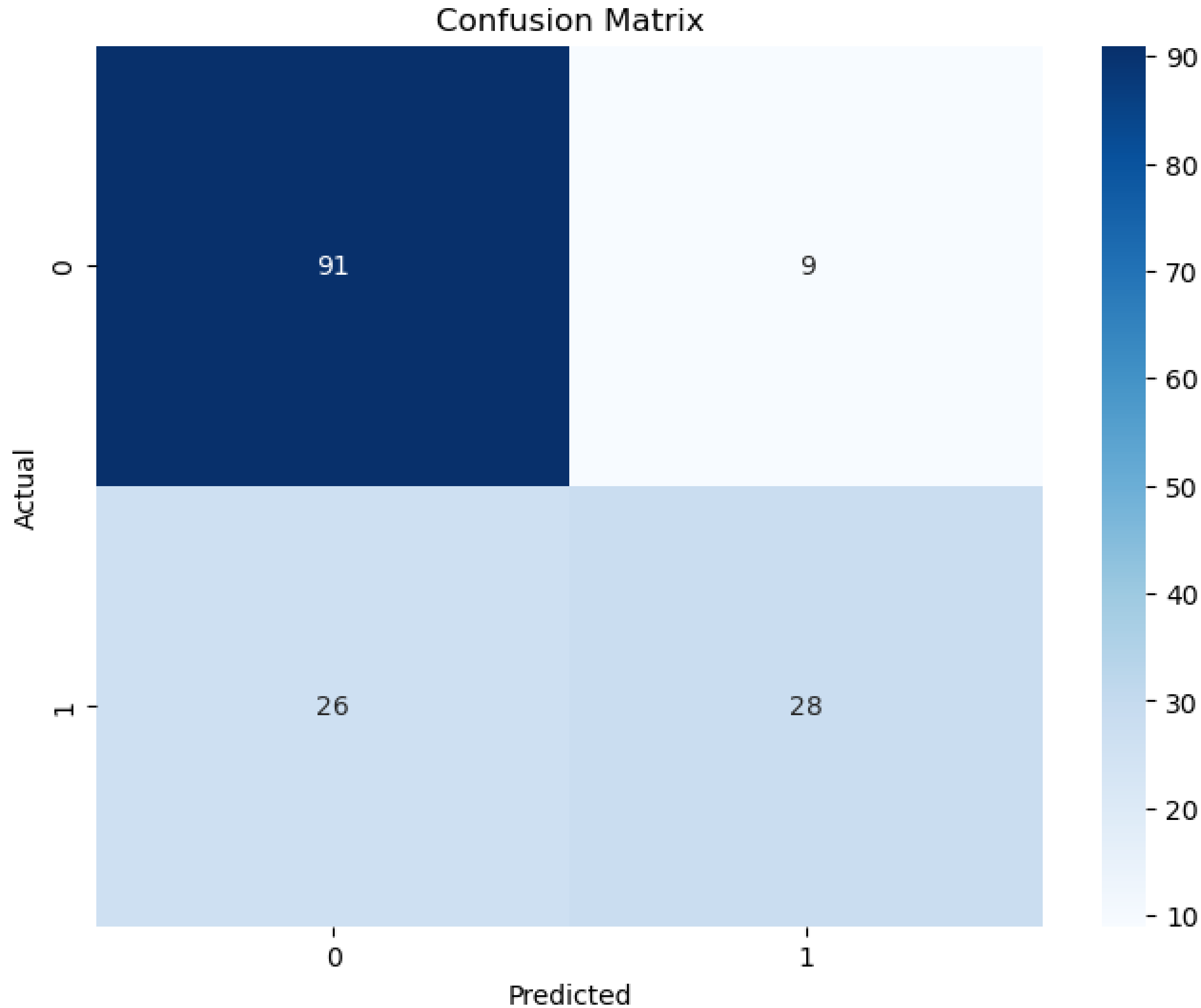
Out[37]:
```
array([0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [38]:
```
import pandas as pd

# Create a DataFrame with y_test and y_pred
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Create the cross-tabulation
cross_tab = pd.crosstab(df['Actual'], df['Predicted'])

print(cross_tab)

plt.show()
```

```
Predicted   0   1
Actual
0          91   9
1          26  28
```

In [39]:

```
# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



3

This is not a good practice:

it is not a good practice to evaluate the accuracy of your model by predicting on the training data itself. The purpose of splitting the dataset into training and testing sets is to evaluate the model's performance on unseen data.

By predicting on the training data (x_train) and comparing the predictions (x_train_preds) with the corresponding true labels (y_train), you are essentially evaluating how well your model fits the training data it has already seen. This does not provide a reliable measure of how well your model will generalize to new, unseen data.

Instead, you should use the separate testing data (x_test and y_test) to evaluate the performance of your model. After training your model using model.fit(x_train, y_train) , you can then use model.predict(x_test) to obtain the predicted labels for the testing data. You can then compare these predictions with the true labels (y_test) to calculate the accuracy on the testing set. This will give you a better indication of how well your model is likely to perform on new, unseen data.

A predictive system can be built with this, based on available data;

In [55]:

```
model.fit(x_train, y_train)
x_train_preds =  model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_preds, y_train)
```

In [56]:

```
x_test_preds =  model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_preds, y_test)
```

Making a predictive system

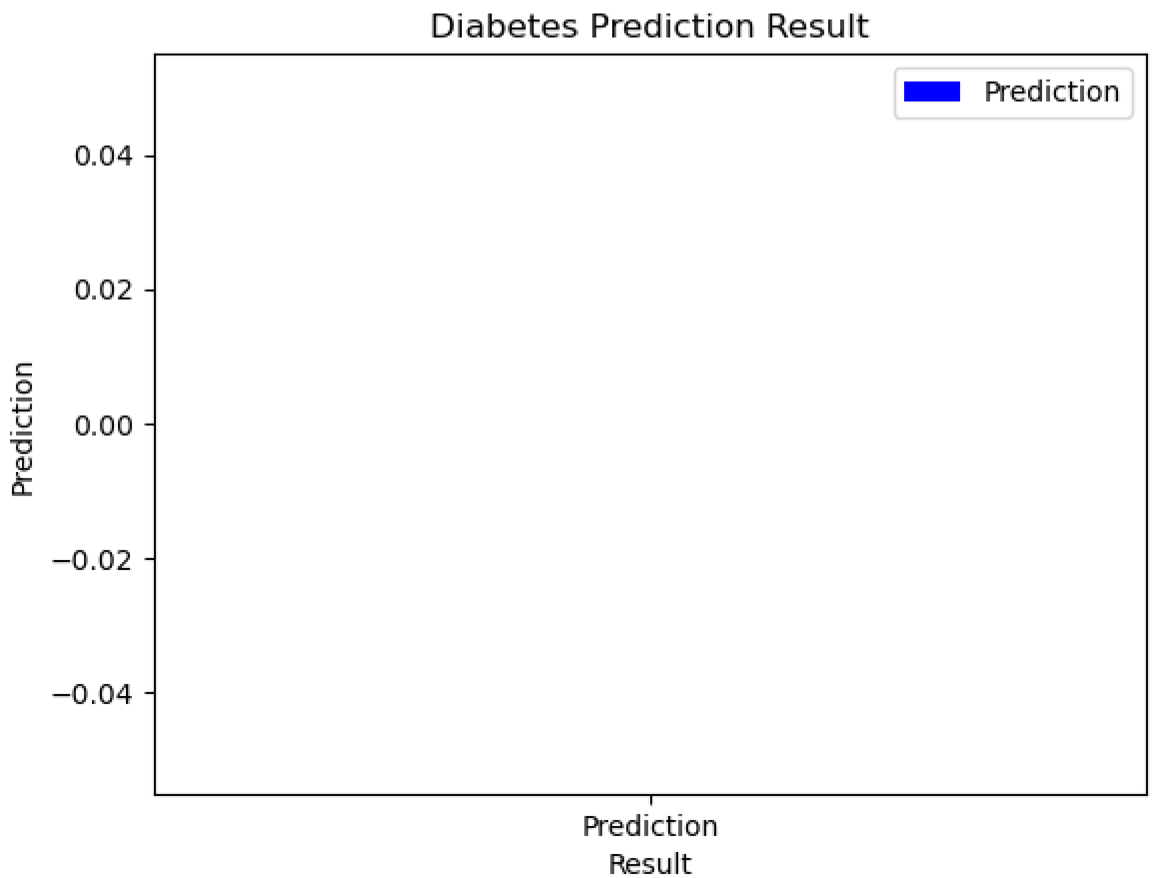
In [69]:
```
input_data = (10,139,80,0,0,27.1,1.441,57,)

# change the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

std = scalar.transform(input_data_reshape)
print(std)


prediction = model.predict(std)
prediction

if prediction[0] == 1:
    print('Diabetics')
else:
    print('Non-Diabetics')
    import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")

```

```
[[ 1.82781311  0.56664949  0.56322275 -1.28821221 -0.69289057 -0.62096232
   2.92686858  2.02160968]]
Non-Diabetics
```

Empty chart means 'Non-Diabetic'

In [80]:

```
input_data = (10, 139, 80, 0, 0, 27.1, 1.441, 57)

# Convert input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

# Scale the data
scaler = StandardScaler()
std = scaler.fit_transform(input_data_reshape)

# Assuming you have the predicted result stored in the 'prediction' variable
prediction = 0  # Replace with your actual prediction value

# Determine the result label
if prediction == 1:
    result = 'Diabetic'
else:
    result = 'Non-Diabetic'

# Plotting the result
plt.bar(['Prediction'], [prediction], color='blue', label='Prediction')
plt.xlabel('Result')
plt.ylabel('Prediction')
plt.title('Diabetes Prediction Result')
plt.legend()
plt.show()
```

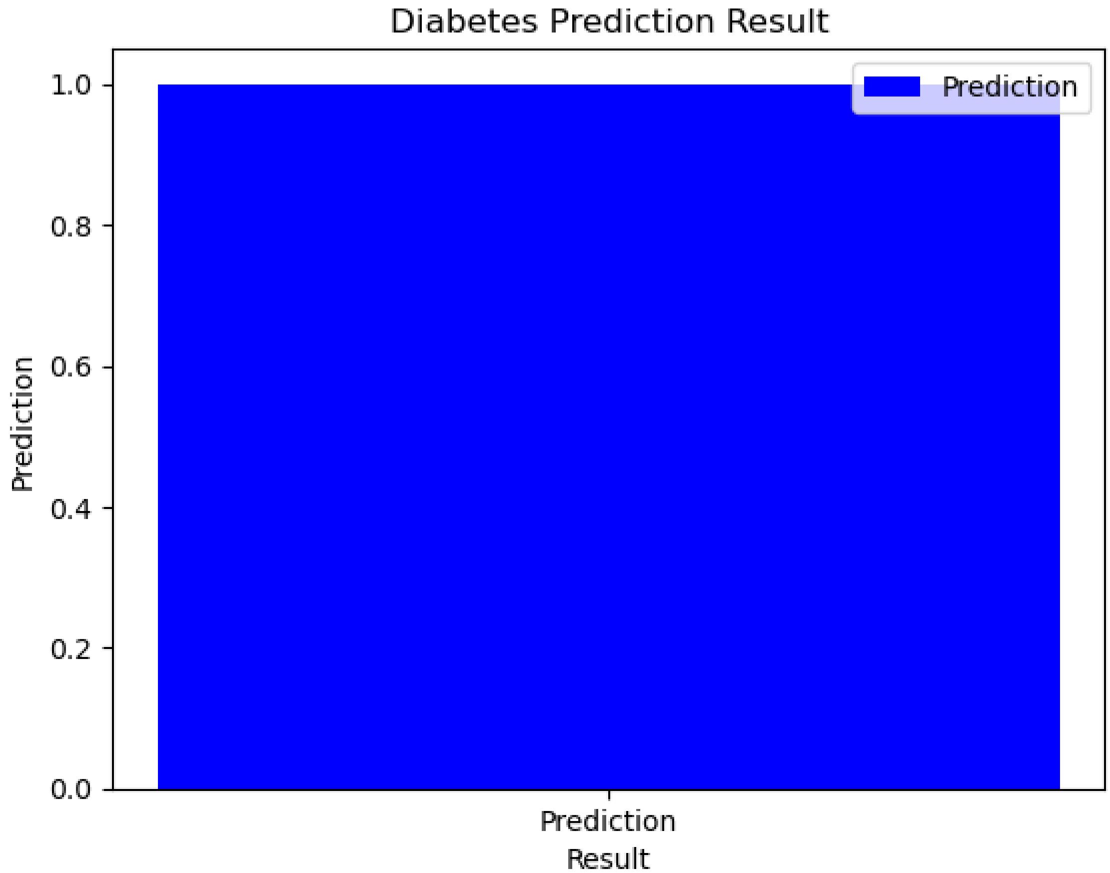

Diabetes Prediction Result
Prediction
0.04
0.02
0.00
−0.02
−0.04
Prediction
Prediction
Result

In [68]:

```
input_data = (1,189,60,23,846,30.1,0.398,59)

# change the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

std = scalar.transform(input_data_reshape)
print(std)


prediction = model.predict(std)
prediction

if prediction[0] == 1:
    print('Diabetics')
else:
    print('Non-Diabetics')
    import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")



```

```
[[-0.84488505  2.13150675 -0.47073225  0.15453319  6.65283938 -0.24020459
  -0.2231152   2.19178518]]
Diabetics
```

In [79]:
```
input_data = (1,189,60,23,846,30.1,0.398,59)

# Convert input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

# Scale the data
scaler = StandardScaler()
std = scaler.fit_transform(input_data_reshape)

# Assuming you have the predicted result stored in the 'prediction' variable
prediction = 1  # Replace with your actual prediction value

# Determine the result label
if prediction == 1:
    result = 'Diabetic'
else:
    result = 'Non-Diabetic'

# Plotting the result
plt.bar(['Prediction'], [prediction], color='blue', label='Prediction')
plt.xlabel('Result')
plt.ylabel('Prediction')
plt.title('Diabetes Prediction Result')
plt.legend()
plt.show()
```


Diabetes Prediction Result
Prediction
1.0
0.8
0.6
0.4
0.2
0.0
Prediction
Prediction
Result

```
1
```